SmartSociety – A Platform for Collaborative People-Machine Computation

Ognjen Scekic*, Daniele Miorandi†, Tommaso Schiavinotto†, Dimitrios I. Diochnos‡, Alethia Hume§, Ronald Chenu-Abente§, Hong-Linh Truong*, Michael Rovatsos‡, Iacopo Carreras†, Schahram Dustdar* and Fausto Giunchiglia§

* Distributed Systems Group, TU Wien, Austria
Email: oscekic | truong | dustdar @dsg.tuwien.ac.at

† U-Hopper, Trento, Italy
Email: daniele.miorandi | tommaso.schiavinotto | iacopo.carreras @u-hopper.com

‡ Centre for Intelligent Systems and their Applications, University of Edinburgh, UK
Email: d.diochnos | mrovatso @inf.ed.ac.uk

§ Department of Information and Communication Technology, University of Trento, Italy
Email: hume | chenu | fausto @disi.unitn.it

***Abstract*—Society is moving towards a socio-technical ecosystem in which physical and virtual dimensions of life are intertwined and where people interactions ever more take place with or are mediated by machines. Hybrid Diversity-aware Collective Adaptive Systems (HDA-CAS) is a new generation of socio-technical systems where humans and machines synergetically complement each other and operate collectively to achieve their goals. HDA-CAS introduce the fundamental properties of hybridity and collectiveness, hiding from the users the complexities associated with managing the collaboration and coordination of machine and human computing elements. In this paper we present an HDA-CAS system called SmartSociety, supporting computations with hybrid human/machine collectives. We describe the platform's architecture and functionality, validate it on two real-world scenarios involving human and machine elements and present a performance evaluation.**

I. Introduction

Over the last few years we have been witnessing a rapidly evolving process of merging of devices and software services with the human society fabric. We are moving more and more towards a world where the physical and virtual dimensions of life become deeply entangled and human interactions ever more often take place with or are mediated by machines. These changes are opening up the possibilities for novel forms of interaction, collaboration and organization of labor. In [1] we analyzed this emerging phenomenon and named it *social collective intelligence* (SCI). Its power resides in the combination of contributions coming from both humans (individuals as well as collectives) and computers. Humans bring in their competences, knowledge, and skills together with networks of social relationships and understanding of social context, while computer elements provide unmatched ability to transfer information over large distances, to store data for long time and to quickly perform well-defined computations at scale. The core idea of SCI is not new. Similar motivation led to the design of different systems, described under the terms: CSCW, Social Computing, Crowdsourcing, and Social Machines. However, most existing systems limit themselves to using computers to support and orchestrate purely human collaborations, usually based on patterns of work that can be predictably modeled before the execution [2]. On the other hand, the concept of social collective intelligence implies blurring the line between human and machine computing elements, and considering them under a generic term of *peers* – entities that provide different functionalities under different contexts, provisioned under a service model [3]; participating in *collectives* – persistent or short-lived teams of peers, representing the principal entity performing the computation (task). Peers and collectives embody the two fundamental properties of the SCI vision: *hybridity* and *collectiveness*, offered as inherent features of these emerging systems. Systems supporting these properties perform tasks and computations transparently to the user by assembling or provisioning appropriate collectives of peers that will perform the task in a collaborative fashion. We call the whole class of these emerging socio-technical systems HDA-CAS[1] [1]. While the general idea of SCI looks appealing at an abstract level, from an engineering perspective building such systems is a challenging task, requiring solutions for problems that go well beyond the traditional coordination and communication problems into the areas of social norms, privacy and ethics [4]. Furthermore, as people and machines are expected to work seamlessly in various collectives and contexts, questions on acceptable means of controlling these collaborations inevitably arise. Both hard/direct (programming, workflows) and soft/indirect means (reputation, incentives) approaches are applicable, but bring along associated technical complexities, drawbacks and ethical issues.

In this paper we present the *SmartSociety Platform*[2], a novel HDA-CAS, able to effectively support a wide spectrum of collaboration scenarios present in today's social computing: from 'open-call' to 'on-demand' collaborations, featuring an advanced coordination and privacy management. Concretely, the contribution of this paper is the design of the platform—description of its functionality, architecture and core com-

[1] Hybrid Diversity-Aware Collective Adaptive Systems, see http://focas.eu/

[2] The platform is being developed in the context of the EU FP7 research project 'SmartSociety' URL:http://www.smart-society-project.eu/

ponents. The paper demonstrates how the platform's design tackles the fundamental HDA-CAS research challenges of hybridity and collectiveness on two real-world case-studies employing a prototype implementation of the platform.

The paper is organized as follows: In Section II we present the intended usage context and design requirements of the platform. In Section III we present the architecture and functionality of the platform. In Section IV we functionally evaluate the platform's support of hybridity and collectiveness through two real-world case studies. In SectionV we report on the implementation and experimental performance evaluation. Related work is described in Section VI. Finally, Section VII concludes the paper and points out directions for future activities.

II. Design Requirements & Usage Context

The SmartSociety platform (platform) is a software framework intended for use by:

1) *Users* – external human clients or applications who need a complex collaborative human-machine task performed;
2) *Peers* – human or machine entities providing Human/Web Services.

The platform acts as intermediary between the two user types, trying to align their interests and provide them the following functionalities:

- For users: *a)* task execution environment; and *b)* workforce management functionality.
- For peers: *c)* a collaboration environment; and *d)* fair working conditions.

While providing these functionalities can be considered as the set of basic design requirements (DR1a–d) for the platform, the distinguishing, novel design requirements assume providing them under the HDA-CAS principles of **hybridity** (DR2) and **collectiveness** (DR3). Throughout the paper, we show how the presented design responds to these requirements.

The intended platform usage context foresees human peers registering their profiles with the platform and enlisting for performing different professional activities. The platform uses this data for locating and engaging peers into different collaborative efforts. Peer engagement is transparent with respect to the working conditions (DR1d): peers know in advance the conditions under which they are required to provide their services, how the effort will be monitored/assessed, as well as what kind of compensation (or penalty) awaits them. In case of human peers, the platform asks for an explicit approval, enabling the peer engagement under a short-term contractual relationship. In case of a software peer, the services are contracted under conventional service-level agreements (SLAs). Once the platform has located appropriate peers for performing a task (computation), a *collective* is formed. A collective is composed of a team of peers along with a collaborative environment assembled for performing a specific task. The collaborative environment (DR1c) consists of a set of software communication and coordination tools. For example, the platform is able to set up a predefined virtual communication infrastructure for the collective members, provide access to a shared data repository (e.g., Dropbox folder) [3], and plan/orchestrate the necessary activities to be performed by the collective's members.

The complete collective lifecycle is managed by the platform (DR1b) in a context of a SmartSociety *platform application*. The applications are encapsulating application-specific business logic that also determines in which way collectives are formed. During the task execution various incentives [5] may be applied for stimulating the collective's effort and retaining the peers. After finishing the task, collectives may be dissolved, and the reputation and other metrics of the member peers are updated (DR1d). The application-specific business logic also determines what is the accepted quality of result (QoR), different adaptation and elasticity policies, metrics and other runtime execution parameters. Furthermore, the application specifies one of the predefined *orchestration* and *negotiation patterns* that the platform enforces during the execution.

Platform users submit task requests to be executed to the SmartSociety platform (DR1a). A *user application* communicates with the corresponding platform application. For example, as shown in Fig. 1, a mobile user application SmartShare contacts the corresponding SmartSociety platform application. Note that the same person can at the same time play the role both of a user and of a peer of a platform application— for example, in a ride-sharing application, a person requests a ride from the platform (as a user), but then takes a part in the ride (e.g., as a driver) and thus plays the role of the peer by providing a service for the platform. Similarly, a same person can participate as a peer in different collectives, in the same or different platform applications concurrently, represented by different peer profiles.

III. Platform Architecture & Functionality

A. Overview

A simplified, high-level view of the SmartSociety platform architecture is presented in Fig. 1. The architecture is designed to be fully distributed and scalable. The rectangle boxes represent the key platform components that may be deployed distributively, as all components expose (private or public) RESTful APIs. The principal component-interoperability channels are denoted with double-headed arrows in the figure. Communication with peers is additionally supported via popular commercial protocols to allow a broader integration with existing communication software and allow easier inclusion of peers into the platform.

User applications contact the platform through the REST API component. All incoming user requests are served by this module that verifies their correctness and dispatches them to the appropriate platform application that will be processing and responding to them. The platform applications run sandboxed in appropriate containers (see Docker[3]), allowing the applications to be deployed at different (virtual) machines. The first time a platform application is run the container will take care of informing the *PeerManager (PM)* component to set up (register) appropriate application, peer and collective profiles required by the application. The container will also request from the PeerManager a set of permissions for accessing and manipulating sensitive private data of peers.

[3]https://www.docker.com/

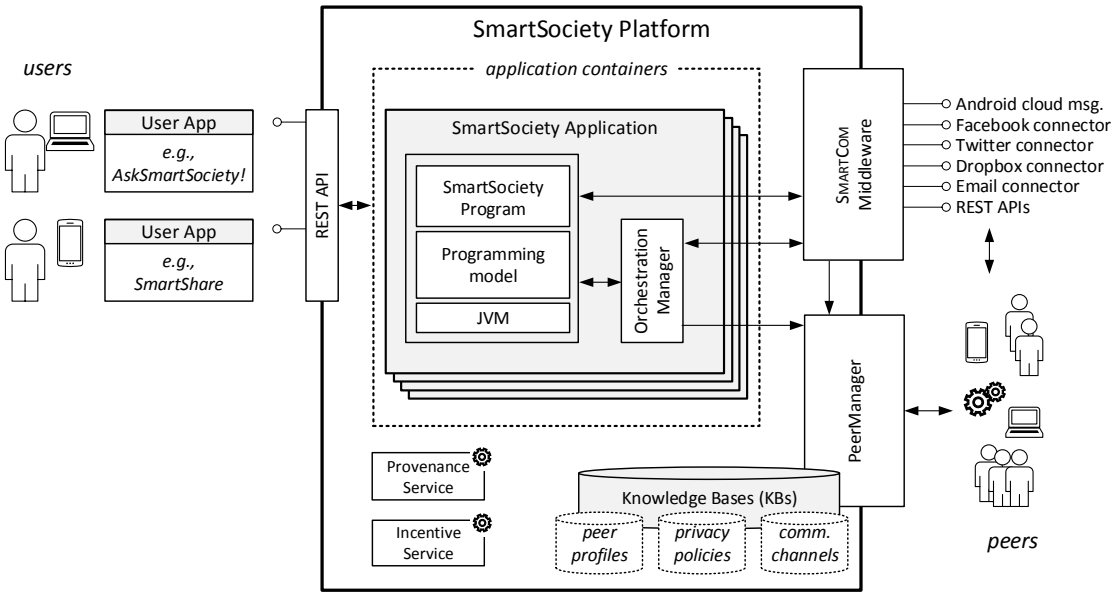

Fig. 1: SmartSociety platform users and architecture.

The PeerManager can shortly be described as the central data-store of the platform, managing all peer and application information, and allowing privacy-aware access and sharing of the data among platform components. In practice, the platform application is a Java application making use of SmartSociety platform's programming libraries, allowing the developer to execute collective-based tasks on the platform. The developer is offered a complete *programming model*[4] and appropriate high-level language constructs. Each platform application features a dedicated *Orchestration Manager (OM)* component. The OM is the component in charge of preparing and orchestrating collaborative activities among peers. Performing these functionalities requires the OM to heavily use the PeerManager and SmartCOM Middleware components.

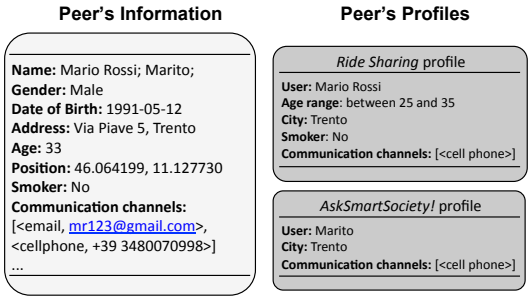

Fig. 2: Simplified example of a peer with multiple profiles. Each profile is revealed to a different application.

B. Principal Components

Peer Manager (PM): provides the central data store that maintains and manages information about human- or machine-based peers in a privacy-preserving framework. Concretely, the PM provides the following functionalities: *a*) A mechanism to manage peers' information (using profiles) that accounts for heterogeneous peers; *b*) A semantic peer search functionality; and *c*) A model for enforcing advanced privacy mechanisms.

To effectively manage peer profiles across different applications, the PM builds upon the notion of an *entity-centric semantic enhanced model* [6] that defines an extensible set of entity schemas providing the templates for an attribute-based representation of peers' characteristics. Concrete meaning of schemas is specified by mapping single elements (i.e., types of entities, names of attributes and their values) to concepts from an underlying ontology that is also part of the same model, thus allowing reasoning over peer's properties as well as the implementation of semantic-enhanced services. The basic set of schemas/templates can be easily extended to support new application-specific attributes, allowing an efficient definition of new peer types. The adaptability is provided by enabling search and information sharing services to work over the new types in a way that is transparent to the rest of the platform.

By leveraging semantic search approaches described in [7] the PM's search functionality allows locating relevant peers and collectives based on a set of attribute constraints even when they are described using different terminology in their profiles. Queries can specify arbitrarily complex semantic operations and constraints on attributes. The semantical search is one of the enabling factors for the overall hybridity property of the platform (DR2), because it allows interpreting queries originating from human peers into a tractable set of constraints, thereby alleviating semantical differences which are inherently present when dealing with humans.

Additionally, the PM defines a *privacy protection model* that pays special attention to different privacy principles enacted by the EU Data Protection Directive 95/46/EC[5] affecting

[4]Description of the programming model is currently in submission preparation and can be made available to the reviewers upon request.

[5]http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:31995L0046

storage and processing of personal data. Specifically, the model defines privacy regulations and considerations described in [8], such as purpose specification and binding, that are enforced upon search queries. This means that in different usage-contexts the peer profiles will reveal only partial or (semantically) obfuscated information, used for replying to specific information requests, thus enforcing data minimization. Fig. 2 shows an simplified example of a human peer subscribed to participate in two platform applications: a ride-sharing application and a Q&A application, revealing different information (by using different profiles) in each case. This allows, e.g., a human peer to reveal its age range (as a way to obfuscate the exact date of birth) when participating in a ride-sharing collective, while the same information is completely hidden when participating in a question-answering collective. More details about the definition, internal design and implementation of the PM can be found here[6].

Communication Middleware: SMARTCOM is a communication and virtualization middleware used as the primary means of communication between the platform and the peers. Although tightly integrated into the platform, SMARTCOM is designed as an independent component that can be used with similar HDA-CAS platforms. Apart from performing functionalities typical of conventional service buses (e.g., message transformation, routing, encryption, authentication) the distinguishing novelty of SMARTCOM is its native support for virtualizing collectives [3]:

- Hiding the complexity of communication with a dynamic collective as a whole and passing of instructions from the HDA-CAS platform to it, making collective a first-class, programmable entity (DR3);
- Making the human vs. machine distinction transparent during the communication, by interpreting/translating the messages for different peer types and delivering them to peers through different communication channels/protocols, in accordance with peer's preferences (DR2);
- Allowing concurrent participation of peers in different collectives concurrently, acting as a different service units with different SLA, delivery and privacy policies.

A peer can use different communication channels to interact with SMARTCOM, e.g., a human peer can communicate with the platform via email and Twitter interchangeably, receive task descriptions and track progress through a web application, and communicate with other peers within the collective through a dedicated mobile app. Human peers can make use of software peers in the collective, serving as collaborative and utility tools. For example, a software service like Doodle can be used to agree upon participation times, or Dropbox as a common repository for performed tasks. The developer also uses SMARTCOM indirectly through the provided language constructs to implement the communication with collectives from the SmartSociety program during the execution phase. PeerManager stores for SMARTCOM the communication and availability preferences of human peers. Before contacting a peer, SMARTCOM checks with the PeerManager possible communication preferences in the peer's profile. Peers can specify: *a)* preferred communication channels (protocol) (e.g., through email or via Twitter); *b)* availability time-slots (e.g., working days 09-17h); *c)* prevent being contacted by specific peers/collectives. When a message needs to be dispatched to a peer, the SMARTCOM takes these preferences automatically into the account, transparently to the rest of the platform. More details on the internal design and functionality of SMARTCOM, as well as the source code is available here[7].

Orchestration Manager (OM): The OM is responsible for the following functionalities:

- *Composition* – Generating possible *execution plans* to meet user-set constraints and optimize wanted parameters.
- *Negotiation* – Coordinating the negotiation process among human peers leading to the overall agreement and acceptance of the suggested execution plan by the participating peers.
- *Execution* – Monitoring the execution and enforcing the selected execution plan during the runtime.

OM works in an asynchronous loop reacting to events of new (users') task requests and (peers') participation requests. Upon each event the OM computes the set of feasible execution plans associated with one or more requests. Plans are constructed by solving a high-level combinatorial or constraint satisfaction problem, as described here[8].

For example, in a ridesharing scenario, drivers post the rides (task requests) and passengers express participation requests (also a type of task requests). Although passengers may be flexible to take a ride in different time intervals during the day, an execution plan can contain only a time interval fitting every participant in the riding collective associated with that plan. Other constraints may need to be considered, such as the capacity of the vehicle; or trade-offs, such as choosing between the optimal route vs. the route that accommodates more participants. In such a setting, each new/altered request can lead to creation/invalidation of multiple plans— e.g., a number of passengers who submitted a participation request could not have been part of any execution plan until a driver submitted the matching ride offer. When the ride is finally offered, multiple possible plans are generated with different passenger collectives, and only one plan can (in this case) be ultimately realized. Furthermore, if at any time the driver cancels the ride, all plans need to be invalidated. Conversely, a matching ride offer by another driver creates a different set of execution plans, opening up a possibility for passengers to concurrently consider and negotiate about participating in different rides, but ultimately choosing only one. Once new plans are generated, the participants in the tentative collectives associated with each plan can negotiate among them, thereby deciding whether the candidate solution provided by the OM is acceptable and the actual execution can take place. OM mediates the negotiation process based on the selected *negotiation protocol (pattern)*. The descriptions of currently supported negotiation protocols are provided here[8]. The OM uses SMARTCOM to enact the negotiation protocol, i.e., to dispatch appropriate offers, accepts, rejects and agreed plans.

Both the negotiation process and the execution process are orchestrated using the *linked data approach*[8], which uses shared artifacts (documents) with unique IDs and versioning

[6]http://www.smart-society-project.eu/publications/deliverables/D_4_2

[7]http://github.com/tuwiendsg/SmartCom

[8]http://www.smart-society-project.eu/publications/deliverables/D_6_2

to trace task requests, plans and status changes of a plan in execution. Such approach allows storing the state of an execution, scalability of the entire process, and provenance generation (Sec. III-C).

The described OM functionalities are fundamental for enabling the human-driven collectiveness (DR3), i.e., workflows where the order of activities is not prescribed (cf. Sec. VI), but is instead determined at runtime, based on the preferences and capabilities of the human peers interested in performing the task.

C. Other Components

The incentive service is an independent component offering an API for storing new incentive mechanisms and subsequently suggesting concrete incentive measures for given peers based on the analysis of their past behavior. Incentives are application-specific, and need to be carefully designed following a case-study analysis. The design of the component was based on the research experiences gained by running a case-study on incentivizing participants in collaborative scientific platform Zooniverse[9] [5]. The service currently supports issuing personalized textual messages to peers, meant to motivate their participation in the collective effort.

Provenance is defined[10] as: "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness". Although designed as an independent component, in our architecture the *Provenance Service (ProvS)* is a component offering a platform-internal server and service for tracing the data provenance from processes involving task executions performed by peers. The provenance traces are stored on ProvS in the graph form [9]. As the same data gets additionally processed by new peers, possibly in the context of new task executions, the provenance graphs get updated. ProvS is a passive component; it requires another platform component to submit the actions performed on a piece of data. In our case, the OM uses the ProvS to trace data related to negotiations and task executions. Any platform component can invoke the service to query provenance graph, and obtain specific results on peer reputation or accountability.

IV. Functional Evaluation

In order to validate the platform's functionality and test its ability to support a wide variety of real-world use cases, we prototyped two SmartSociety platform applications and the corresponding peer/user applications and ran case-studies with actual users/peers. The experience gained from running the two case-studies provided us the feedback information which led to subsequent design improvements. The two applications implement the two extremes of the collaborative man-machine computation spectrum— a design choice made for evaluating and showcasing the flexibility of the platform.

The first one, called **SmartShare** is a ride-sharing application. In terms of collaboration pattern, SmartShare can be considered an *open-call* application. In 'open-call' collaborations tasks are published on the platform, and peers are motivated

[9]http://www.zooniverse.org/

[10]W3C Definition: http://www.w3.org/TR/prov-overview/

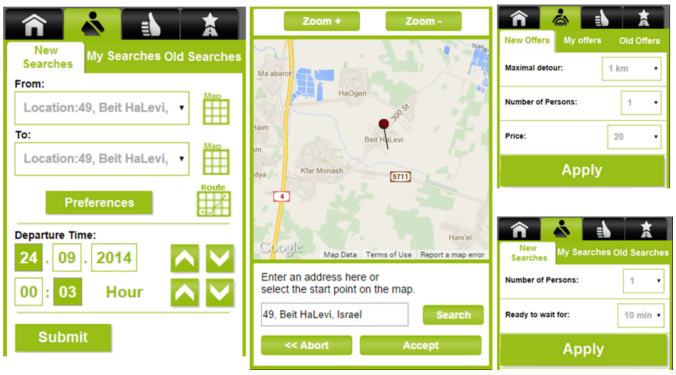

Fig. 3: Partial screenshots of the SmartShare user+peer mobile application.

to apply and negotiate for taking part in execution of the task. Typical representatives of 'open-call' collaborations are today's micro-task and design contest crowdsourcing platforms [10]. However, they lack advanced composition and negotiation features presented here. The second one, called **AskSmartSociety!**, is a collaborative question and answer (Q&A) service able to combine responses from individuals, collectives and machines. In terms of collaboration pattern, AskSmartSociety! can be considered an *on-demand* application. In 'on-demand' collaborations, for each input task the platform tries to locate or provision peers/collectives that should be optimally capable of performing the task with respect to given input constraints (e.g., [11]).

SmartShare: SmartShare is a ride-sharing application in which travelers share a vehicle for a trip and split travel costs such as gas, toll, and parking fees with others that have compatible itineraries and time schedules. In SmartShare the platform performs the following functionalities: *a)* generating ride plans by matching compatible ride offers and ride requests (composition); *b)* providing recommendations to peers based on their preferences and the reputation of other peers; *c)* orchestrating negotiation process among human peers.

On the other hand, human peers are in charge of: *d)* posting and searching for rides; *e)* accepting and withdrawing from potentially agreed rides as other travelers sign up or better opportunities arise; *f)* participating in the ride either as drivers or as passengers; and *g)* rating each other after the completion of the ride.

A single individual can play concurrently the role of both a platform user (by posting ride requests/offers) and a peer (by taking a ride). The ride requests and offers are posted through web or mobile user application interfaces, that play the role of the joint user+peer application (see Fig. 3). The request includes information on the date, time and destination, as well a set of user preferences (e.g., "no smoking during the ride"). The platform then goes on to mediate the negotiation process, in which each peer identified in the composed plan is again contacted over the preferred communication channel in line with personal privacy settings (cf. Sec. III-B) and asked to express interest in taking the ride. If all intended peers provide a confirmation, the ride is fully agreed and execution can start. However, until all peers have agreed, peers are only in conditional acceptance, and able to withdraw it at any time.

This allows the peers to contend for multiple rides in parallel, and finally agree to a single one or none. The described negotiation procedure represents just one possible negotiation pattern that the platform is able to support. The execution phase is not actively mediated by the platform, but rather driven by the collective participating in the ride. Indirect (soft) techniques are used to control the execution, such as feedback-based peer reputation and incentives (achievement badges and encouragement emails, such as in [5]). The reputation is calculated through the ratings of drivers and passengers left after the ride, and subsequently used by the OM when performing the matching.

In the period late 2014–early 2015 a pilot run of the SmartShare application took place in Israel, involving students and staff of the Ben Gurion University. The application accumulated 150 registered users/peers, out of which 84 close successful ridesharing agreements through the application. Running this case study allowed us to test the full spectrum of possible composition and negotiation scenarios in a realistic environment (DR3), as well as to improve the design and application of incentives, provenance and reputation. The complete case-study of the pilot run can be found here[11]. The SmartShare codebase can be accessed here[12].

AskSmartSociety!: AskSmartSociety! is a Q&A service where tasks submitted by the users correspond to natural-language questions that need to be transparently answered by hybrid collectives consisting of both humans and/or software services such as Google or Twitter. Differently than other Q&A services such as Yahoo! Answers or StackOverflow.com, where questions are openly posted for peers to respond, AskSmartSociety! is able to actively (on-demand) locate and engage those individuals or software services which can be expected to provide answers of required quality. Quality criteria determine the composition of the collective, i.e., different service matching and ranking. For example, if response speed is the preferred criterion, then aggregated answers from different software-based peers will be presented to the user. On the other hand, if human interpretation of the question's context is required, or local knowledge, then human peers can filter the responses from software services, or provide their own answers.

Consider the following scenario: Bob is about to visit Milan for the first time. He is looking for a nearby restaurant with a garden, that will be showing the game of his favorite team. In addition, Bob requires gluten-free meals. The application is able to respond to such questions by leveraging hybrid collectives which rely on software services to come up with a list of restaurants with garden in Bob's vicinity, but rely on local human peers to filter out those who have gluten-free meals and showing games and rank/recommend them. Answers can be ranked based on the reputation of the peers or community ranking (similarly to StackOverflow). In some instances the user issuing the question can select an answer and provide quality feedback to the peers.

In the current implementation stage, the AskSmartSociety! application is able to include into answering collectives Google and Twitter as two machine peers. While Google is accessed directly through the proper API endpoint, Twitter is leveraged through SMARTCOM's Twitter connector. This allows us to post tweets with questions and automatically harvest answers decorated with a specific hashtag. Human peers are accessible through a dedicated peer Android application interfaced through SMARTCOM, or indirectly, using the Twitter peer as mediator.

The work on AskSmartSociety! is still at an early stage, so no full-scale user study was run. However, experiences gained from selected users (staff and students of the University of Trento) allowed us to test assembly and management of collectives of various levels of hybridity (DR2). The AskSmartSociety! codebase can be accessed here[13]. Also a video showcasing the basic end-to-end functionality is available here[14].

V. PERFORMANCE EVALUATION

A. Prototype Implementation

The platform is comprised of a set of independent components implemented in different technologies, communicating through a set of RESTful APIs. Application Runtime is implemented as a lightweight Java application based on a Jetty standalone server that manages and executes the platform applications provided by the developer. It exposes a RESTful interface that allows the user application to submit tasks to the platform applications and monitor them. It also handles calls to other components on behalf of the platform applications. SMARTCOM's core is implemented in Java, as well as the provided adapters that allow the message delivery over different channels/protocols. SMARTCOM internally uses Apache Active MQ as an industry-standard message broker. The Orchestration Manager is implemented as a NodeJS web application, internally relying on MongoDB for temporary task storage. Peer Manager consists of a back-end written in Java using PostgreSQL for storage, and a NodeJS front-end exposing a higher-level API. Incentive and Provenance Service are implemented as independent Python/Django web applications.

The platform supports both multi-instance and multi-tenant deployment models. Platform components (e.g., OM and SMARTCOM) can be replicated and partitioned per application and run in a Docker container, allowing flexible scaling. Alternatively, a single OM and SMARTCOM instance can handle multiple compatible platform applications in parallel, sharing resources and peers.

B. Experimental Evaluation

The platform's scalability and performance was evaluated for 1, 5, 10, 20, 50 and 100 SmartSociety platform applications, simulated as Java worker threads (denoted as 'Worker' in Fig.4), sending $1 \cdot 10^6$ messages concurrently, uniformly distributed to 1, 10, 100, and 1000 peers waiting for messages and replying to them. Each test run was executed 10 times to obtain average throughput results. The simulation was made on a machine with the following specifications: Windows 7 64-bit, Intel Core2 Duo @ 2.53 GHz, 4.00 GB RAM.

[11] http://www.smart-society-project.eu/publications/deliverables/D_5_3

[12] https://gitlab.com/smartsociety/orchestration

[13] https://gitlab.com/smartsociety/appruntime.

[14] https://youtu.be/Jr9z2Coqc6M

Figure 4 presents the results of the test runs. The average throughput remains between 5000 and 3000 msg/sec. The performance decrease with higher amounts of peers is the result of increased memory requirements rather than computational complexity. The limiting factor here is the ActiveMQ message broker used in the SMARTCOM implementation which only allows for a maximum of approximately 20,000 msg/sec. The platform has an upper bound of 5,000 msg/sec since each message is handled multiple times by the broker. This limitation applies to a single SMARTCOM instance, but multiple instances can be deployed on the platform to balance the load if needed, sharing the PeerManager access. Communication performance and peer scalability is thus not expected to become a concern due to the increased latency of human peers and variance of response times compared to machine peers.

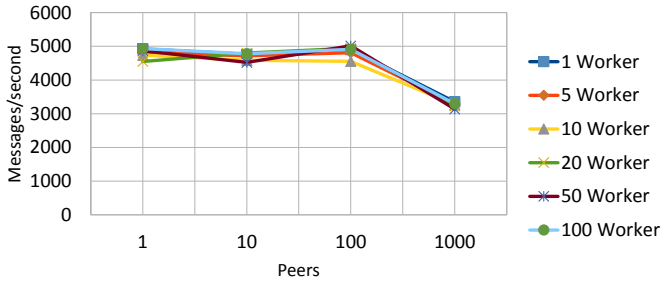

Fig. 4: Simulated message throughput for [1, 100] platform applications (denoted as 'Worker') communicating bidirectionally with [1, 1000] peers concurrently.

Another potentially critical performance bottleneck is the performance of computationally-expensive composition algorithms or the time-consuming synchronization that is potentially needed during negotiation. To test this, we performed two experiments designed to examine the robustness and scalability of the OM in a simulated ride-sharing scenario. In the experiments, artificial "groups" of size k in a population of n agents (representing human peers) were created, such that all the ride requests inside a group would match and the number of generated plans could be controlled. Up to 10 groups of 6, 9, and 12 agents each were created, were the passenger agents were twice the size of the number of driver agents in each group. The OM generated all possible plans in every case, i.e., $(2^p - 1) \cdot d$, where $p/d \in \{4/2, 6/3, 8/4\}$, allowing up to 10,200 plans generated in the extreme case.

All the drivers would pick the ride plan with the largest amount of passengers, and then all the passengers would simultaneously agree to participate in a ride that they could potentially agree to. Eventually, in every group the collective would be formed by one driver and all the passengers. However, in the last agreement that would arrive to the system, the agreed ride plan had to be finalized.

In the first experiment (with the described setup) composition took on average less than a minute in the case of 120 total agents and negotiation for all 10 groups of the same case was accomplished in less than 1.5 minutes. In the second experiment, artificial delays were introduced for the communication of the agents. This could simulate delays between the different operations that agents perform in the real world. The results indicate that when the delay increases by a factor of 10, the overall lifespan of an agent increases only by a factor of 3 to 4. Thus there is a good indication that the system scales well, as communication delays are expected to be much bigger in a real-world scenario, due to the asynchronous nature in which humans will process and negotiate on tasks.

VI. RELATED WORK

Due to space constraints here we present an overview of relevant classes of socio-technical systems, their typical representatives, and compare their principal features with the SmartSociety platform. Based on the way the workflow is abstracted and encoded the existing approaches can be categorized into three groups: *a*) programming-level approaches; *b*) parallel-computing approaches; and *c*) process modeling approaches.

Programming level approaches focus on developing a set of libraries and language constructs allowing general-purpose application developers to instantiate and manage tasks to be performed on socio-technical platforms. Unlike SmartSociety, the existing systems do not include the design of the crowd management platform itself, and therefore have to rely on external (commercial) platforms. The functionality of such systems is effectively limited by the design of the underlying platform. Typical examples of such systems are TurKit [12] , CrowdDB [13] and AutoMan [14]. TurKit is a library layered on top of Amazon's Mechanical Turk offering an execution model ("crash-and-rerun") which re-offers the same microtasks to the crowd until they are performed satisfactorily. The entire synchronization, task splitting and aggregation is left entirely to the programmer. On the other hand, and unlike SmartSociety, the inter-worker synchronization is out of programmer's reach. The only constraint that a programmer can specify is to explicitly prohibit certain workers to participate in the computations. CrowdDB similarly outsources parts of SQL queries as mTurk microtasks (e.g., let the crowd provide a value for a data field or sort data rows). AutoMan integrates the functionality of crowdsourced multiple-choice question answering into Scala programming language. The authors focus on automated management of answering quality. The answering follows a hardcoded workflow. Synchronization and aggregation are centrally handled by the AutoMan library. The solution is of limited scope, targeting the designated labor type. Neither of the systems allows explicit collective formation, or hybrid collective composition.

Parallel computing approaches rely on the divide-and-conquer strategy that divides complex tasks into a set of subtasks solvable either by machines or humans. Typical examples include Turkomatic [15] and Jabberwocky. For example, Jabberwocky's [16] *ManReduce* collaboration model requires users to break down the task into appropriate map and reduce steps which can then be performed by a machine or by a set of humans workers. Hybridity is supported at the overall workflow level, but individual activities are still performed by homogeneous teams. In addition, the efficacy of these systems is restricted to a suitable (e.g., MapReduce-like) class of parallelizable problems. Also, in practice they rely on existing crowdsourcing platforms and do not manage the workforce independently, thereby inheriting all underlying platform limitations.

The process modeling approaches focus on integrating human-provided services into workflow systems, allowing modeling and enactment of workflows comprising both machine and human-based activities. They are usually designed as extensions to existing workflow systems, and therefore can perform certain peer management. The two currently most advanced systems are CrowdLang [17] and CrowdComputer [2]. CrowdLang brings in a number of novelties in comparison with the previously described systems, primarily with respect to the collaboration synthesis and synchronization. It enables users to (visually) specify a hybrid machine-human workflow, by combining a number of generic (simple) collaborative patterns (e.g., iterative, contest, collection, divide-and-conquer), and to generate a number of similar workflows by differently recombining the constituent patterns, in order to generate a more efficient workflow at runtime. The use of human workflows also enables indirect encoding of inter-task dependencies. Even if CrowdLang allows a certain level of runtime workflow adaptability, it is limited to patterns that need to be foreseen at design-time. SmartSociety differs from both of these systems mostly by extending the support for collaborations spanning from processes known at design-time to fully human-driven, ad-hoc runtime workflows. CrowdComputer is a platform allowing the users to submit general tasks to be performed by a hybrid crowd of both web services and human peers. The tasks are executed following a workflow encoded in a BPMN-like notation called BPMN4Crowd, and enacted by the platform. CrowdComputer can be seen as the platform resembling most closely the functionality offered by the SmartSociety platform. However, while CrowdComputer assumes splitting of tasks and assignment of single tasks to individual workers through different 'tactics' (e.g., marketplace, auction, mailing list) SmartSociety natively supports actively assembling hybrid collectives to match a task. In addition, by providing a programming abstraction, SmartSociety offers a more versatile way of encoding workflows.

VII. Conclusion

In this paper we presented SmartSociety – an HDA-CAS platform supporting collaborative computations performed by hybrid collectives, composed of software and human-based services. The platform is able to host user-provided applications, implementing and managing on their behalf a spectrum of collaborative patterns, ranging from on-demand to open-call. Unlike existing solutions, the platform allows executing runtime/ad-hoc, human-driven workflows. It offers advanced automated composition of viable execution plans, coordination of the negotiation process, transparent and multifaceted virtualization and communication with peers/users via different protocols, and privacy-aware management of peer/user profiles. The platform's design has been influenced by the two presented real-world case studies, designed to cover the two extremes of the collaborative spectrum and used to validate the novel HDA-CAS properties of the platform— hybridity and collectiveness.

Future work will see the completion and tighter integration of all project-developed components into a unified framework. In addition, it foresees the development of cognate collaborative platform applications under the broadly defined 'tourism' subject area, which should help validate and improve the current design. Talks are currently under way to test the two existing applications in real, unsupervised environments in municipalities of Northern Italy and Israel.

Acknowledgment

This work is supported by the EU FP7 SmartSociety project under grant No. 600854.

References

[1] D. Miorandi, V. Maltese, M. Rovatsos, A. Nijholt, and J. Stewart, Eds., *Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society*. Springer, 2014.

[2] S. Tranquillini, F. Daniel, P. Kucherbaev, and F. Casati, "Modeling, enacting, and integrating custom crowdsourcing processes," *ACM Trans. Web*, vol. 9, no. 2, pp. 7:1–7:43, May 2015.

[3] P. Zeppezauer, O. Scekic, H.-L. Truong, and S. Dustdar, "Virtualizing communication for hybrid and diversity-aware collective adaptive systems," in *ICSOC 2014 workshops and satellite events*, ser. WESOA'14. Springer, 11 2014, p. Forthcoming.

[4] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *Proc. of the 2013 Conf. on Computer supported cooperative work*, ser. CSCW '13. ACM, 2013, pp. 1301–1318.

[5] A. Segal *et al.*, "Improving productivity in citizen science through controlled intervention," in *Proc. of WWW'15*, 2015, pp. 331–337.

[6] F. Giunchiglia, B. Dutta, and V. Maltese, "From knowledge organization to knowledge representation," in *ISKO UK Conference*, 2013.

[7] F. Giunchiglia and A. Hume, "A distributed entity directory," in *The Semantic Web: ESWC 2013 Satellite Events*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 7955, pp. 291–292.

[8] M. Hartswood, M. Jirotka, R. Chenu-Abente, A. Hume, F. Giunchiglia, L. A. Martucci, and S. Fischer-Hübner, "Privacy for peer profiling in collective adaptive systems," in *Privacy and Identity Management for the Future Internet in the Age of Globalisation*. Springer, 2015.

[9] L. Moreau, "Aggregation by provenance types: A technique for summarising provenance graphs," in *GRAPHS AS MODELS 2015*, vol. 181. Electronic Proceedings in Theoretical Computer Science, February 2015, pp. 129–144.

[10] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Comm. ACM*, vol. 54, no. 4, pp. 86–96, 2011.

[11] B. Sengupta, A. Jain, K. Bhattacharya, H.-L. Truong, and S. Dustdar, "Who do you call? problem resolution through social compute units," in *Proc. 10th Intl. Conf. on Service-Oriented Comp.*, ser. ICSOC'12. Springer-Verlag, 2012, pp. 48–62.

[12] G. Little, "Exploring iterative and parallel human computation processes," in *Ext. Abstracts on Human Factors in Comp. Sys.*, ser. CHI EA '10. ACM, 2010, pp. 4309–4314.

[13] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: Answering queries with crowdsourcing," in *Proc. 2011 ACM SIGMOD Intl. Conf. on Management of Data*, ser. SIGMOD '11. ACM, 2011, pp. 61–72.

[14] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "Automan: A platform for integrating human-based and digital computation," *SIGPLAN Not.*, vol. 47, no. 10, pp. 639–654, Oct. 2012.

[15] A. P. Kulkarni, M. Can, and B. Hartmann, "Turkomatic: Automatic recursive task and workflow design for mechanical turk," in *CHI '11 Ext. Abs. on Human Factors in Comp. Sys.*, ser. CHI EA '11. ACM, 2011, pp. 2053–2058.

[16] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabberwocky programming environment for structured social computing," in *Proc. 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. ACM, 2011, pp. 53–64.

[17] P. Minder and A. Bernstein, "Crowdlang: A programming language for the systematic exploration of human computation systems," in *Social Informatics*, ser. LNCS, K. Aberer, A. Flache, W. Jager, L. Liu, J. Tang, and C. Guéret, Eds. Springer, 2012, vol. 7710, pp. 124–137.